PRESENTING DATA

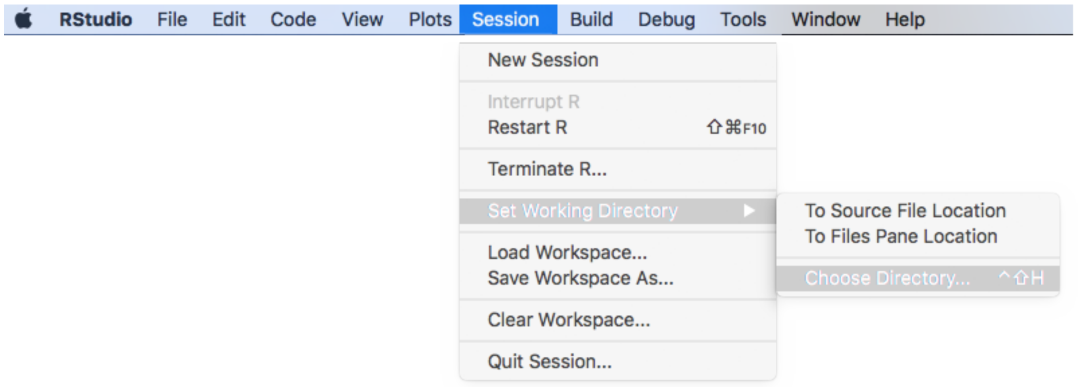
This handout is one of a series that accompanies *An Adventure in Statistics: The Reality Enigma* by me, Andy Field. These handouts are offered for free (although I hope you will buy the book).[1]

Overview

In this handout we will look at how to do the procedures explained in Chapter 5 using R an open-source free statistics software. If you are not familiar with **R** there are many good websites and books that will get you started; for example, if you like *An Adventure In Statistics* you might consider looking at my book *Discovering Statistics Using R*.

Some basic things to remember

- RStudio: I assume that you're working with RStudio because most sane people use this software instead of the native **R** interface. You should download and install both R and RStudio. A few minutes on Google will find you introductions to R Studio in the event that I don't write one, but these handouts don't particularly rely on R Studio except in setting the working directory (see below).
- Dataframe: A dataframe is a collection of columns and rows of data, a bit like a spreadsheet (but less pretty)
- Variables: variables in dataframes are referenced using the $ symbol, so *catData$fishesEaten* would refer to the variable called *fishesEaten* in the dataframe called *catData*
- Case sensitivity: **R** is case sensitive so it will think that the variable *fishesEaten* is completely different to the variable *fisheseaten*. If you get errors make sure you check for capitals or lower case letters where they shouldn't be.
- Working directory: you should place the data files that you need to use for this handout in a folder, then set it as the working directory by navigating to that folder when you execute the command accessed through the **Session>Set Working Directory>Choose Directory ...** menu



[1] This document is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, basically you can use it for teaching and non-profit activities but not meddle with it.

Packages used in this chapter

We install packages using the *install.package("package name")* function, so if you haven't already got them installed execute the following commands:

```
install.packages("ggplot2")
install.packages("ggthemes")
install.packages("scales")
```

This installs the package **ggplot2**, which we use to plot graphs. In the Chapter Zach is taught about Edward Tufte's guidelines for presenting data, the package **ggthemes** gives us access to a Tufte-inspired theme for ggplot2 to use when generating plots, and the package **scales** is needed to plot percentage data such as in pie charts. Once installed we need to initialise these packages in the current **R** session by executing the *library(packagename)* function as follows.

```
library(ggplot2)
library(ggthemes)
library(scales)
```

The data

In the book, Zach has attended a recruitment event for JIG:SAW in which Rob Nutcot has presented various graphs. Rob's graphs are horrible and Milton takes Zach to visit Dr. Tuff who teaches him how to present data properly. This handout looks at how to create graphs that would keep Dr Tuff more or less happy. (We will recreate the Dr Tuff friendly graphs from the book chapter.) The data set is too big to enter manually so instead we will read the data in from the csv file on the companion website for the book. To do this, execute the command below, which uses the *file.choose()* function to open a dialog box so that you can navigate to the file that you want to open, which in this case will be *AiS Ch 05 JIGSAW Data.csv*. The rest of the command tells R to import this file into a dataframe called *jigsaw*

```
jigsaw<-read.csv(file.choose())
```

We can use the *head()* function to look at the top of the data file. Executing the command below will display the top 10 lines of the **jigsaw** dataframe. If you want to look at a different number of lines then change the number 10, and if you want to see the whole data then execute the name of the dataframe (**jigsaw**) rather than using the *head()* function.

```
head(jigsaw, 10)
##    ID         Employee   Job_Type Footspeed Strength    Vision    Sex
## 1   1 JIG:SAW Employee Scientists     13.88 1161.000 1.3444883 Female
## 2   2 JIG:SAW Employee Scientists     17.34 1141.000 0.3899555   Male
## 3   3 JIG:SAW Employee Scientists     14.06 1174.000 0.7710037 Female
## 4   4     Non-Employee Scientists     21.84 1320.974 0.6463387   Male
## 5   5 JIG:SAW Employee Scientists     15.53 1112.000 0.3478982   Male
## 6   6     Non-Employee Scientists     16.27 1153.050 0.4058432   Male
## 7   7 JIG:SAW Employee Scientists     14.52 1185.000 0.3562858 Female
## 8   8 JIG:SAW Employee Scientists     12.22 1095.000 0.7325945 Female
## 9   9     Non-Employee Scientists     16.59 1071.825 0.5139766 Female
## 10 10     Non-Employee Scientists     13.88 1217.723 0.4381180 Female
```

Looking at the top 10 cases you will see that the dataframe contains 7 variables:

- **ID**: identifies the participant.
- **Employee**: codes whether a participant worked for JIG:SAW or not (this variable is known as a factor).

- **Job_Type**: codes the type of job the participant had (all of the top 10 were scientists but if you look at the whole data you will see other job descriptions).
- **Footspeed**: this variable contains the participants' footspeed (mph).
- **Strength**: this variable contains the participants' maximal push force (N).
- **Vision**: this variable contains the participant's visual acuity scores.
- **Sex**: this variable codes the participant's biological sex as male or female.

As a slight inconvenience, the graphs involving the variable **Sex** in the book are ordered with Male as the first category and female as the second. When R reads the data in it creates factors (variables coding group membership) alphabetically, which means that the order of categories for **Sex** will be Female then Male: the opposite way around to the book. This difference will confuse/annoy some people, and if you're one of those people you should remove the potential stress and re-order the variable **Sex** by executing this command.

```
jigsaw$Sex<-factor(jigsaw$Sex, levels(jigsaw$Sex)[c(2, 1)])
```

This command uses the *factor()* function to recreate the **Sex** variable in the **jigsaw** dataframe. The first term in the function (*jigsaw$Sex*) tells it to create *jigsaw$Sex* from itself, and the second term (*levels(jigsaw$Sex)[c(2, 1)]*) tells it to set the levels of the variable to be the same as those for *jigsaw$Sex* but in the order 2, 1. In other words it reverses the order from Female, Male to Male, Female.

ggplot 2

We are going to create all of the graphs in this tutorial using Hadley Wickham's ggplot2 package because it's awesome. However, it is so awesome that there is a lot to explain. Too much to get into here. I have a lengthy chapter in my book Discovering Statistics Using R, and Hadley has his own book which I think is really good (I believe a second edition is due 2016). For this tutorial I will simply show you the code for the graphs in my book *An Adventure in Statistics* and explain what each bit does without getting into the underlying architecture of ggplot2.

Bar graphs

To obtain a bar chart of mean strength by the person's biological sex and whether or not they worked at JIG:SAW we could execute the following R code. Note that resulting graph matches Dr Tuff's graphs in Figure 5.4 in *An Adventure in Statistics*.

```
strengthBar <- ggplot(jigsaw, aes(Employee, Strength, fill = Sex))

strengthBar + stat_summary(fun.y = mean, geom = "bar", position = "dodge") + stat_summary(fun.data
= mean_sdl, geom = "pointrange", position = position_dodge(width = 0.90)) + labs(x = "Employee Stat
us", y = "Mean Maximal Push Force (N)")  + coord_cartesian(ylim = c(0, 3000)) + scale_y_continuous(bre
aks = seq(0, 3000, 200)) + theme_tufte() + scale_fill_brewer(palette = "YlGnBu")
```

There's a lot to unpick here. The first line creates an object called *strengthbar* using the *ggplot()* function. Within this function we specify the dataframe that we want to use (**jigsaw**), and then within the *aes()* function we specify the variable that we want on the *x*-axis (**Employment**), the variable we want on the *y*-axis (**Strength**), and the variable we want to fill with different colours (**Sex**). This line of code creates the basic graph object, we now need to add things to it.

The second block of commands takes the basic graph object and adds layers to it. Let's break this command down piece by piece:

- *strengthBar +*: tells R to take the object *strengthBar*, which we created in the first command and add stuff to it. The following commands specify different things that we are adding (hence all of the + symbols)
- *stat_summary(fun.y = mean, geom = "bar", position = "dodge")*: This command uses the *stat_summary()* function to create bars on the graph. First we tell it that we want to use the function 'mean', which will give us the mean values of whatever we're plotting on the y-axis (*fun.y = mean*), then we tell it to display the mean using bars (*geom = "bar"*), finally we tell it that we don't want to allow these bars to overlap (*position = "dodge"*). This final instruction is important because we've asked (when we created the object *strengthBar*) to use the variable **Sex** to produce different bars for males and females, so we don't want these overlapping.
- *stat_summary(fun.data = mean_sdl, geom = "pointrange", position = position_dodge(width = 0.90))*: This command uses the *stat_summary()* function to add error bars to the graph. First we tell it that we want to use the function 'mean_sdl', which will give us the standard deviation values of whatever we're plotting on the *y*-axis (*fun.data = mean_sdl*), then we tell it to display the standard deviation using a vertical line (*geom = "pointrange"*), finally we again use dodge to make sure that the error bars don't clash on the horiozontal axis, and set the width (*position = position_dodge(width = 0.90)*).
- *labs(x = "Employee Status", y = "Mean Maximal Push Force (N)")*: This command adds labels to the *x*- and *y*-axes. Note that the labels need to be in speech marks.
- *coord_cartesian(ylim = c(0, 3000))*: This command sets the limits of the *y*-axis to be 0 and 3000.
- *scale_y_continuous(breaks = seq(0, 3000, 200))*: This command sets the *y*-axis to run from 0 to 3000 in intervals of 200.
- *theme_tufte()*: this function applies a template form the **ggthemes** package that is based on Tufte's principles of data visualisation.
- *scale_fill_brewer(palette = "YlGnBu")*: This command sets the fill colours to be the same as the book - it overrides the default fill colours.

Saving graphs

If you want to save the graph (and this applies to subsequent graphs in this handout), you can use the *ggsave()* function. You specify the name of the file that you want to use in speech marks. You must remember to include the file extension because ggplot2 uses this file extension to determine the format to save in; I have used .png to save as a PNG file but you can save as lots of different formats including the widely used .pdf, .jpeg, .tiff, and .bmp. The other terms in the function specify that I want to save a file that is 7 wide by 5 high, and the units of measurement is inches. This will produce an image 7 × 5 inches, will be a good size for this image. The image will be saved in the current working directory with the name given in the function (in this case *JIGSAW bar chart.png*).

```
ggsave("JIGSAW bar chart.png", width = 7, height = 5, units = "in")
```

Line graphs

To produce the same graph but using lines rather than bars we could execute the following R code. Note that resulting graph matches Dr Tuff's graphs in Figure 5.5 in *An Adventure in Statistics*.

```
strengthLine <- ggplot(jigsaw, aes(Employee, Strength, colour = Sex))

strengthLine + stat_summary(fun.y = mean, geom = "point", size = 4) + stat_summary(fun.y = mean, ge
om = "line", aes(group= Sex)) + labs(x = "Employment", y = "Mean Maximal Push Force (N)", colour = "
Sex") + coord_cartesian(ylim = c(0, 2000)) + scale_y_continuous(breaks = seq(0, 2000, 500)) + theme_tuf
te()
```

The first line is exactly the same as before except we've changed the name of the object we're creating to be *strengthLine* and we're asking that the colour rather than the fill be varied by biological sex (this is because bars on a bar chart can have a fill colour, but lines cannot). The rest is quite similar to with the bar chart so I'll point out the differences and you can compare and contrast.

- *strengthLine +*: again tells R to take the object *strengthLine*, which we created in the first command and add stuff to it.
- *stat_summary(fun.y = mean, geom = "point", size = 4)*: This command specifies that we want the mean but note now that instead of bars we have asked for dots (*"point"*) and specified the size of these dots to be 4.
- *stat_summary(fun.y = mean, geom = "line", aes(group= Sex)*: This command again specifies that we want the mean but this times sets the geom to be a line (which will connect the dots added by the previous command). We have also specified that we want separate lines for different categories of biological sex (*aes(group= Sex)*).
- *labs(x = "Employment", y = "Mean Maximal Push Force (N)", colour = "Sex")*: This command adds labels to the *x*- and *y*-axes, but also for the colour aesthetic, which means that the legend telling us which colour represents which biological sex will be labelled *Sex*.
- *coord_cartesian(ylim = c(0, 2000))*: This command sets the limits of the *y*-axis to be 0 and 2000.
- *scale_y_continuous(breaks = seq(0, 2000, 500))*: This command sets the *y*-axis to run from 0 to 2000 in intervals of 500.
- *theme_tufte()*: same as before.

Boxplots (Box-Whisker diagrams)

Zach also showed Dr Tuff a graph of footspeed by whether or not the person was employed by JIG:SAW (Figure 5.6 in the book). To produce the version that Dr Tuff produced, we can execute the following commands.

```
footBox <- ggplot(jigsaw, aes(Employee, Footspeed))
footBox + geom_boxplot() + labs(x = "Employee Status", y = "Footspeed (mph)") + theme_tufte()
```

The first line is similar to before. We create a plot object called *footBox* and then within the *ggplot()* function we specify the dataframe to be used (**jigsaw**) and within the *aes()* function specify the variable that we want on the *x*-axis (**Employment**), and the variable we want on the *y*-axis (**Footspeed**).

- *footBox +*: again tells R to take the object *footBox*, which we created in the first command and add stuff to it.
- *geom_boxplot()*: This command specifies that we want a boxplot of the data.
- *labs(x = "Employee Status", y = "Footspeed (mph)")*: This command adds labels to the *x*- and *y*-axes.
- *theme_tufte()*: applies a Tufte-inspired theme.

Scatterplots

Subsetting data

Zach showed Dr Tuff a graph of footspeed against strength in people employed at JIG:SAW. Therefore, we want to work only with the data from JIG:SAW employees. We can do this by creating a new dataframe that is a subset of the whole dataframe using the *subset()* function. This command creates a new dataframe called *employees*. It does this using the *subset()* function, within which we specify the data frame that we want to subset (**jigsaw**) and a logical argument that tells **R** what cases to retain. In this case I have used *Employee == "JIG:SAW Employee"*, which means that if the value of the variable **Employee** is equal to *"JIG:SAW Employee"* then we keep it in the new dataframe. The resulting dataframe will be the same as the **jigsaw** dataframe except that it will only contain cases for whom the value of the variable **Employee** is is equal to *"JIG:SAW Employee"*. Use *head()* again to look at the dataframe.

```
employees<-subset(jigsaw, Employee == "JIG:SAW Employee")
```

Scatterplots

Having created a new dataframe containing only the JIG:SAW employees, we can use this to plot the scatterplot that Dr Tuff produced (Figure 5.8), by executing the following commands.

```
scatter <- ggplot(employees, aes(Footspeed, Strength))

scatter + geom_point(shape = 16, size = 4, alpha = 0.5) + geom_smooth(method = "lm", fill ="lightblue
", alpha = 0.4) + labs(x = "Footspeed (mph)", y = "Maximal Push Force (N)") + coord_cartesian(ylim = c
(0, 5000)) + scale_y_continuous(breaks = seq(0, 5000, 500))+ theme_tufte()
```

The first line is similar to before. We create a plot object called *scatter* and then within the *ggplot()* function we specify the dataframe to be used (**employees**), and within the *aes()* function specify the variable that we want on the *x*-axis (**Footspeed**) and the variable we want on the *y*-axis (**Strength**).

- *scatter +*: again tells R to take the object *scatter*, which we created in the first command and add stuff to it.
- *geom_point(shape = 16, size = 4, alpha = 0.5)*: This command specifies that we want to plot data as points, we specify a shape for those points (*shape = 16* will give us circles), a size. Importantly I have set the points to be semi-transparent by setting alpha to 0.5 (50% transparency). This is because there is a lot of overlap in the data points and having each point as semi-transparent means that we;'ll be able to see these areas of overlap as darker circles.
- *geom_smooth(method = "lm", fill ="lightblue", alpha = 0.4)*: This command plots the line on the graph using the method 'lm' (which stands for linear model). This basically plots a linear regression line and a confidence interval around it. I've specified the shading of this confidence interval to be lightblue with 40% transparency (*alpha = 0.4*) so that it doesn't obscure the data.
- *labs(x = "Footspeed (mph)", y = "Maximal Push Force (N)")*: This command adds labels to the *x*- and *y*-axes.
- *coord_cartesian(ylim = c(0, 5000))*: This command sets the limits of the *y*-axis to be 0 and 5000.
- *scale_y_continuous(breaks = seq(0, 5000, 500))*: This command sets the *y*-axis to run from 0 to 5000 in intervals of 500.
- *theme_tufte()*: applies a Tufte-inspired theme.

Pie charts

Dr Tuff will baste your face with butter if you attempt a pie chart, however, like Milton did in the book you can create a bar chart of percentage data (Figure 5.10 in the book), by executing the following commands. For this to work you need to have referenced the package *scales* so make sure you have executed *library(scales)*.

```
jobHistogram <- ggplot(employees, aes(Job_Type))
jobHistogram + geom_bar(aes(y = (..count..)/sum(..count..)), fill = "lightblue") + scale_y_continuous(labels = percent_format()) + labs(x = "Job Type", y = "Percent") + theme_tufte()
```

We're again interested in only the JIG:SAW employees so use the **employees** dataframe. The first line is similar to before. We create a plot object called *jobHistogram*, and then within the *ggplot()* function we specify the dataframe to be used (**employees**), and within *aes()* specify the variable that we want on the *x*-axis (**Job_Type**).

- *jobHistogram +*: again tells R to take the object *jobHistogram*, which we created in the first command and add stuff to it.
- *geom_bar(aes(y = (..count..)/sum(..count..)), fill = "lightblue")*: This command specifies that we want to plot data as bars, we specify that we want to plot the count divided by the sum of counts, which gives us the proportion. We specify to fill the bars in the colour *lightblue*.
- *scale_y_continuous(labels = percent_format())*: this command sets the y-axis to express the proportions from the previous command as percentages.
- *labs(x = "Job Type", y = "Percent")*: This command adds labels to the *x*- and *y*-axes.
- *theme_tufte()*: applies a Tufte-inspired theme.

Closing notes

The possibilities with ggplot2 are endless and we have only scratched the surface, but hopefully this handout gives you a flavour of the power that ggplot2 has and gets you started with looking at more detailed resources.

This handout is written to be used in conjunction with: Field, A. P. (2016). An adventure in statistics; the reality enigma. London: Sage.